Introducción a Docker

Creación de app fullstack de ejemplo

¿Qué es docker?

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos

¿Qué es docker?

Docker es una especie de emulador de programas aislados.

Docker consta de imágenes y contenedores.

Una imagen es una parte mínima y suficiente del sistema operativo para ejecutar programas.

Un contenedor es un entorno aislado con la copia de una imagen la cual se puede configurar.

Instalación de Docker en ubuntu

1. Instalamos docker y docker-compose:

```
#   sudo snap install docker docker-compose
```

2. Añadimos al usuario de ubuntu al grupo docker

```
#   sudo usermod -aG docker $USER
```

3. En https://hub.docker.com/ podemos buscar imagenes de docker ya preparadas

4. Podemos comprobar qué imágenes tenemos con:

```
#   sudo docker images
```

Comandos básicos de Docker

```
#    docker images
```

```
#    docker container ls    o     # docker ps
```

```
#    docker pull IMAGEN
```

Comandos básicos de Docker

```
#   docker build
#   docker run IMAGEN
#   docker inspect CONTENEDOR
#   docker logs CONTENEDOR
#   docker exec -it CONTENEDOR /bin/bash
#   docker start / stop / restart CONTENEDOR
```

Cómo crear un contenedor Docker

1. Crear el directorio de trabajo
2. Entrar en el directorio de trabajo
3. Crear el fichero Dockerfile
4. Ejecutar docker build

Cómo crear un contenedor Docker

Ejemplo:

```
#    mkdir docker_example
#    cd docker_example
#    docker pull nginx
#    docker images
#    docker ps
#    docker run -d --name "web" -p 80:80 nginx;
#    docker ps
#    docker logs [CONTENEDOR]
#    docker inspect [CONTENEDOR]
```

Pasando código local a un contenedor Docker

Ejemplo:

```
#    mkdir www
#    cd www
#    echo “<h1>MI CONTENEDOR NGINX CON DOCKER</h1>” > index.html
#    cd ..
#    docker stop web
#    docker ps
#    docker run -d --name "web" -p 80:80 -v $(pwd)/www:/usr/share/nginx/html nginx;
#    docker ps
#    docker ps -a
#    docker rm [CONTENEDOR]
#    docker ps -a
```

Pasando código local a un contenedor Docker

Ejemplo:

```
#    docker run -d --name "web" -p 80:80 -v $(pwd)/www:/usr/share/nginx/html nginx
#    docker ps
```

- Comprobamos http://localhost en un navegador
- Modificamos index.html
- Comprobamos http://localhost en un navegador

EJERCICIO

Clonar un componente y hacer que se muestre la demo en nginx

Haciendo más cosas con Docker

¿ Y si necesito crear un servidor express en node para exponer un api ?

Vayamos por partes.

1. Vamos a crear un contenedor con node, express y mongodb.
2. Vamos a crear un contenedor con un servidor de mongodb
3. Vamos a conectar los dos contenedores

Servidor con node, express y mongodb

```
# mkdir api (dentro de docker_example)
# cd api
# npm init
# npm install --save express mongodb
```

Servidor con node, express y mongodb

Vamos a crear un fichero index.js con un servidor express mínimo:

```
const express = require('express');
const app = express();
const PORT = 3000;
app.get('/', function(req, res) {
    res.json({"hello": "express with mongo"});
});
app.listen(PORT, function(){
    console.log('Your node js server is running on PORT:',PORT);
});
```

Ejecutamos:

```
# node index.js
```

Probamos http://localhost:3000

Comandos de Dockerfile

FROM nos permite especificar desde qué imagen base de Docker Hub (https://hub.docker.com/) queremos construir.

RUN nos permite ejecutar un comando.

WORKDIR establece un directorio como el directorio de trabajo para las instrucciones COPY, RUN y CMD.

COPY permite copiar archivos o un directorio completo desde una fuente fuera del contenedor a un destino dentro del contenedor.

EXPOSE expone el puerto en el que el contenedor escuchará.

CMD establece el comando predeterminado para ejecutar nuestro contenedor.

Nuestro Dockerfile para el api node-express

Creamos el fichero *Dockerfile* con la configuración de nuestra imagen:

```
FROM node:8
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8080
CMD [ "node", "index.js" ]
```

Fichero .dockerignore

Creamos el fichero .*dockerignore* para evitar copiar a la imagen los ficheros que no queramos copiar en ella.

```
node_modules
*.log
docker-compose.yml
Dockerfile
data
```

Construyendo y corriendo el contenedor node

Construimos la imagen según le indica Dockerfile:

```
# docker build -t manufosela/api .
```

```
# docker images
```

Lanzamos el contenedor

```
# docker run -p 3000:3000 -d manufosela/api
```

```
# docker ps
```

Pregunta

¿Si cambio algo en el index.js se refleja en http://localhost:3000?

¿Por qué?

Solución

¿Cómo podemos hacer para que cualquier cambio en mi local se refleje en el servidor node-express?

```
# docker run -p 3000:3000 -d -v $(pwd)/index.js:/usr/src/app/index.js manufosela/api
```

Modificamos index.js

Aún no se ven los cambios reflejados http://localhost:3000

```
# docker restart [CONTENEDOR_ID]
```

Ahora sí se ven reflejados los cambios en http://localhost:3000

Docker de mongoDB

```
#    docker pull mongo
#    docker images
#    docker run -it -d mongo
#    docker ps
#    docker run -it --link=[NOMBRE_CONT]:mongo mongo /bin/bash
#    env
```

Nos fijamos en la ip

```
#    mongo [IP]:27017
```

Probamos mongodb

```
#    show dbs
#    use local
#    show collections
#    db.startup_log.find({})
#    db.startup_log.find({}).pretty()
#    use mydbdb.mycollection.insert({elemento:"uno"})
#    show collections
#    db.mycollection.find()
```

Conectar docker node con docker MongoDB

Para conectar el contenedor de node, express, mongodb con el contenedor de MongoDB vamos a valernos de docker-compose

Docker-compose es un orquestador de contenedores para que se relacionen entre ellos.

Se configura mediante un archivo .yml llamado *docker-compose.yml*

En dicho fichero se indica qué contenedores se enlazan con quien, de manera que de una sola llamada podemos arrancar y relacionar varios contenedores.

Creamos el fichero docker-compose.yml

```
version: "2"
services:
  app:
    container_name: app
    restart: always
    build: .
    ports:
      - "3000:3000"
    links:
      - mongo
  mongo:
    container_name: mongo
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"
```

Probando docker-compose

Paramos todos los contenedores anteriores de mongo y api

```
#    docker build
#    docker-compose up -d
#    docker-compose ps
#    docker ps
```

Podemos probar que tenemos servidor de node y de mongo.

Podemos parar todos los contenedores de una vez.

```
#    docker-compose down
#    docker-compose ps
#    docker ps
```

Usando mongo en nuestro API

Modificamos el fichero index.js añadiendo lo que viene en negrita:

```
const express = require('express');
const app = express();


const mongodb = require('mongodb');
const config = {
  DB: 'mongodb://mongo:27017'
};

const PORT = 3000;
app.get('/', function(req, res) {
    res.json({"hello": "express with mongo"});
});


var dbo;
const client = mongodb.MongoClient;
```

Usando mongo en nuestro API

```
client.connect(config.DB, function(err, db) {
    if(err) {
        console.log('database is not connected')
    }
    else {
        console.log('connected!!');
        dbo = db.db("midb");
    }
});
app.get('/misdatos', function(req, res){
    let data = dbo.collection("micoleccion").find({}).toArray((err, result) => {
        if (err) throw err;
        res.json(result);
      });
});
app.listen(PORT, function(){
    console.log('Your node js server is running on PORT:',PORT);
});
```

Usando mongo en nuestro API

Ahora para que los cambios tengan efecto debemos parar todo, volver a construir los contenedores y volver a lanzarlos:

```
#    docker-compose down
#    docker-compose build
#    docker-compose up -d
#    docker-compose ps
```

Para poder probar el api debemos cargar con datos la base de datos llamada mibd y la colección de esa base de datos micoleccion

Probamos los entry-point

Mediante la consola de mongo creamos la base de datos:

```
#    mongo
     > use midb
     > db.micoleccion.insert({“titulo”: “primero de prueba”})
     > exit
```

Probamos en http://localhost:3000/misdatos